TASK AND MESSAGE PRIORITY ASSIGNMENT IN AUTOMOTIVE SYSTEMS

Michaël Richard, Pascal Richard, Francis Cottet

Laboratory of Applied Computer Science, ENSMA
BP 40198 Téléport 2
F-86960 Futuroscope
{*richardm,richardp,cottet*}*@ensma.fr*

Abstract: We study in this paper the priority assignment problem for a multiple fieldbus computer architecture in automotive systems. In such systems, tasks and messages are on-line scheduled according to fixed priorities given by the designers. We present a branch and bound algorithm that automatically searches a set of priorities so that all tasks will meet their deadlines. The schedulability analysis is based on the computation of the worst-case response times of the tasks and the messages. The method uses the holistic analysis, that has been modified in order to be integrated in our branch and bound. The method has been applied to an industrial case: the multiple networked in-vehicle embedded system. Numerical results are presented and show the efficiency of the presented approach for complex distributed real-time systems.

Keywords: Multiple Fiedlbus Architecture, Fixed-Priority Scheduling, Priority Assignment, Branch and Bound, Automotive Distributed Systems.

1. INTRODUCTION

The use of communication networks in modern factories or in embedded real-time applications is rapidly increasing. Although the communication device adds a critical resource to the system, the distributed approach for a real-time application is the main key to handle the complexity of a large system and to improve fault tolerance in safety critical applications (Kopetz, 1997). These applications are composed of tasks distributed on different nodes, that communicate by exchanging messages across a communication device; no common memory is available. In such application domains, a distributed hard real-time application is an application that must return coherent results in a fixed delay, namely a strict timing constraint. All application tasks must meet their deadlines, and the exchanged messages have to be sent in a bounded delay in order to be consumed by the destination tasks in time. Particularly, in the area of embedded process automation, the most stringent demands have their origin in the requirements of the control loops. The duration of the real-time transaction between the sensor nodes and the driver nodes is an uppermost important parameter for the quality of control.

Under this framework, the used networks, named fieldbuses, have been developed with the specific requirements of tight real-time capabilities (Thomesse, 1999). Fieldbuses have to strive to respect deterministic response times. Prominent European fieldbus examples, targeted for automotive real-time applications, are the CAN (Controller Area Network) and the VAN (Vehicle Area Network) (ISO, 1994*a*; ISO, 1994*b*). Both correspond to the medium access control protocol, based on the CSMA/CA (Carrier Sense Multiple Access Collision Avoidance) protocol. Although, our work can be applied to other real-time communication protocols, we focused this presentation on these two fieldbuses, used in the considered example car application. In some cases, the maximum communication capacity can be reached or the application, including both critical and non-critical system functions, must ensure that non-critical nodes cannot affect the correct operation of critical ones. To solve these problems, a node can be transformed into a gateway node to open a way to a new fieldbus; so we obtain a multiple network architecture. This solution offers a much more communication potential and/or is able to support a separation between critical and none critical communications.

Schedulability analysis of hard real-time distributed systems can only be done using a worst-case analysis since no necessary and sufficient schedulability condition is known for fixed-priority-driven scheduling of such systems. When priorities are a priori known, the schedulability analysis is usually based on the computation of the worst-case response times of the tasks and the messages in order to guarantee the end-to-end deadlines (Buttazzo, 1997). In that way, the holistic analysis (Tindell and Clark, 1994; Sjödin and Hansson, 1998; Richard *et al.*, n.d.) leads to a sufficient schedulability condition for distributed systems with totally defined schedulers on each processor and each network. The priorities are input data of the holistic analysis when a fixed priority scheduler is considered.

An important issue is the assignment of the priorities to the tasks and the messages in order to enforce the schedulability of the system. Choosing a set of priorities for a small system is a simple job and does not required any computation tool. But for large system, as such considered in today's automotive systems, defining the set of priorities becomes very difficult. Bad choice leads to an unschedulable system that enforces the designers to overdimension components. Consequently, the cost of the system increases and that is not acceptable in the automotive industry. Several heuristics have been proposed in the literature for solving this problem (Gutierrez-Garcia and Gonzalez-Harbour, 1995; Tindell *et al.*, 1992; Silva and Fraga, 2000; Dipippo, 2001). But from the best of our knowledge, no optimal method has been proposed in the literature. Next we propose such a method, based on the holistic analysis as a schedulability test. So, in the remainder of the paper we limit to "holistic schedules" (i.e., schedules that will be validated by the holistic analysis). Our algorithm always find a set of priorities for the tasks and the messages, if a feasible holistic schedule exists.

The rest of the paper is organized as follow: in section 2, we present the industrial application that will be used throughout the text. The section 3 presents the branch and bound algorithm that optimally assigns priorities to the tasks and the messages. Section 4 presents the computational results. Section 5 summarizes the conclusions of the paper and indicates some future works.

2. COMPUTER ARCHITECTURE OF AUTOMOTIVE SYSTEMS

Even if introducing real-time techniques in vehicle industry is still rare (Norström *et al.*, 2000), nowadays car manufacturers integrate more and more micro-controllers that manage the breaks, the injection, the performance, and the passenger fitness (Cavalieri *et al.*, 1996; Castelpietra *et al.*, 2000; Song *et al.*, 1999). For instance, the aims of the engine control system are to manage the engine performance in terms of power, to reduce consumption and to control the emission of exhaust fumes. This control is obtained by sending computed values to the actuators: electronic injectors, electromagnetic air valve for managing idling state of the engine (i.e. the driver do not accelarate) and fuel pump. The ABS system prevents the wheels from blocking when the driver brakes. The system must also take into account sudden variation in the road surface. This regulation is obtained by reading periodically the rotation sensors on each front wheel and this one for the back wheels (usually measured from the differential). If a wheel is blocked, then the ABS system directly acts on the brake pressure actuator. Table 1 describes the functions of the different processors. Complementary information on the process control functionalities can be found, for instance, in (Cavalieri *et al.*, 1996).

These processors are interconnected with different fieldbuses such as CAN and VAN. In order to improve reusability of software components, the processors run common operating systems, as OSEK/VDX (Osek, 1997). Such approach drastically reduces the software development costs. The operating system defines task modules that are scheduled by a fixed priority scheduler. These priorities are given by the designers of software. In the same way, the messages sent in the networks CAN and VAN are scheduled according to fixed priorities, defined by the designers. Figure 1 presents the software architecture and relations among tasks considered throughout the paper (Castelpietra *et al.*, 2000). The software has 44 tasks distributed among the nine processors and 19 different messages go through the two networks.

Fixed-priority and static scheduling can be used in this field of applications (Lonn and Axelsson, 1999). In static scheduling, a pre-run time scheduler calculates a cyclic schedule that is in-

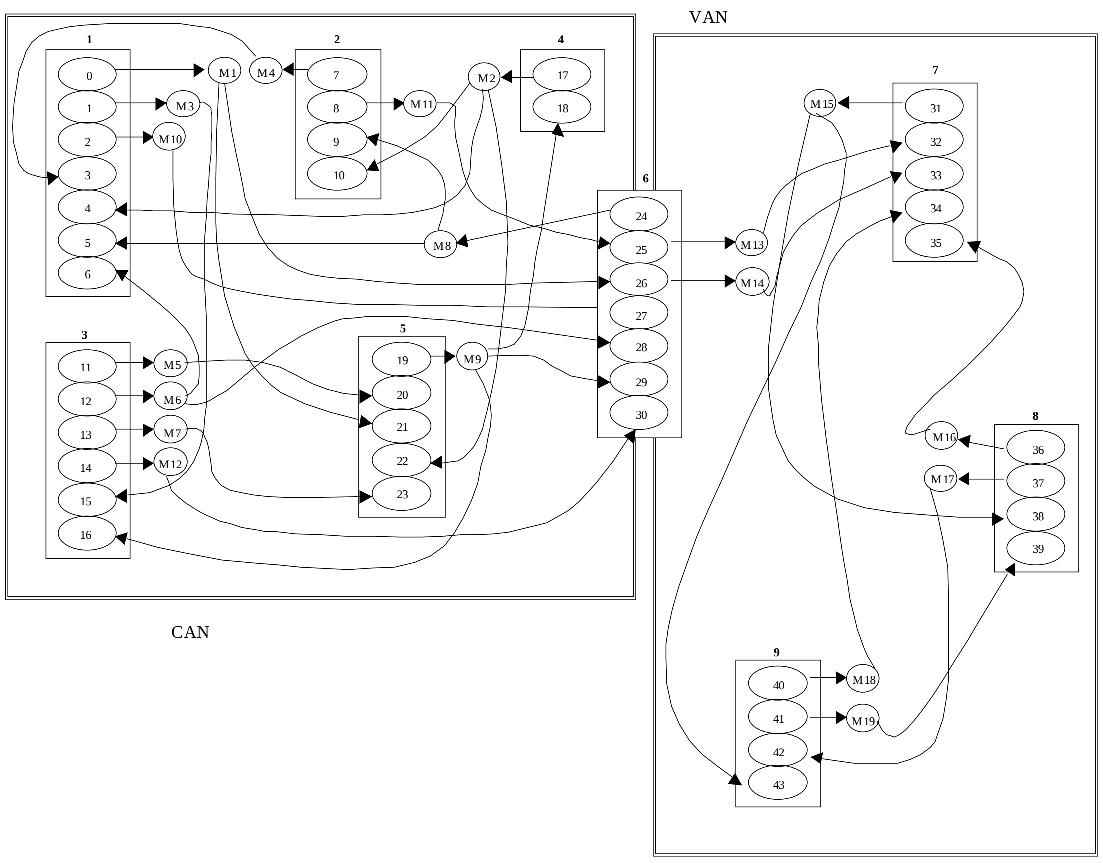

Fig. 1. Software architecture of the application

Processor sites	**Number**
Engine Controller	1
Automatic Gear Box	2
Anti-Blocking Brake System/ Vehicle Dynamic Control	3
Wheel Angle Sensor/ Dynamic Headlamp Corrector	4
Suspension controller	5
Bodywork	6
X	7
Y	8
Z	9

Table 1. Number of each processor

finitely repeated at run-time. The pattern of this cyclic schedule is stored in tables (one per processor or network), that is used by the run-time dispatchers. This approach can be extended in order to handle interrupts (Sandström *et al.*, 1998). The first drawback is the size of the table that is directly related to the least common multiple of the task periods (but, methods have been proposed in order to limit this problem (Cavalieri *et al.*, 1995; Liu, 2000)). Secondly, such scheduling approach is based on a strong synchronization assumption of the physical clocks of the processors. The third problem is that commercial software components do not incorporate such dispatching strategies. Such problems are not encountered in fixed-priority systems. Furthermore, the operating system OSEK/VDX, used in automotive industry, has a fixed-priority scheduler. That is why we only focus on fixed priority scheduling.

We assume that tasks are periodic, preemptive and synchronously released at the beginning of the application (i.e., at time 0). Each task τ_i is defined by three parameters:

- C_i is the worst-case execution time.
- D_i is the relative deadline to the arrival of the task.
- T_i is the period between two successive arrivals of the task in the system.

The communications of the periodic tasks generates periodic traffic on the networks. A message is modelled like a task by three parameters. So, for a given message i, C_i is the worst-case propagation delay of the message and the deadline D_i and the period T_i are inherited respectively of its receiver and sender. Without loss of generality, we next assume that deadlines are not greater than the periods in order to simplify the mathematical relation of the worst-case response times. In this embedded vehicle application, the first arrival time is fixed to zero for every task. The mapping of the tasks on the processor is static and known before the beginning of the application. For a processor, or network, numbered p there are n_p tasks mapped on it. So the total number of tasks and messages is $n = \sum n_p$.

From the schedulability point of view, networks are viewed as additional processors and messages as new non-preemptive tasks with completion times defined by the worst-case propagation delays. In next sections, we neither distinguish tasks and messages, nor networks and processors. From this model of the software system, the communication constraints are modeled with simple precedence constraints among distributed tasks.

3. ASSIGNING PRIORITIES TO TASKS AND MESSAGES

We present a branch and bound algorithm that solves the priority assignment problem. The search of a solution is performed with a search tree that stores all possible permutations to priority assignments to the tasks. Each vertex in the tree assigns a priority to a given task. The assignments are done processor by processor and every underlying sub-trees are separated by a fictitious root vertex. Processors and networks are order in the search tree in increasing order of the workload of their mapped tasks. Such ordering is in fact an heuristic since any ordering can be implemented, but it affects the performance of the method (i.e., the number of backtracks). For each processor, the priorities are assigned from the lower one to the higher one. So, the levels of a sub-tree define the priority levels of tasks on the corresponding processor. The global structure of the search tree is presented in figure 2. When a goal vertex is reached, each task has been assigned a priority on a processor. The algorithm stops when such assignment is validated by the holistic analysis. As consequence, the method limits the search to feasible holistic schedules.

Table 2 presents the pseudo-code of the branch and bound algorithm. The unexplored vertices are stored in the Active Set A. The root vertex of a sub-tree (dedicated to a given processor) is initialized with all possible tasks eligible to the lowest priority level. We now describe the different rules that define the branch and bound search strategies.

• The vertex selection rule selects the next vertex to be explored. Different strategies are commonly used in the literature: First-In-First-Out (FIFO) and Last-In-First-Out (LIFO). The LIFO strategy performs a depth-first search in the tree, FIFO a breadth-first search. Since breadth first search focus on searching a solution in the lowest possible vertex level, it is inadequate for solving our problem (complete solutions are always in goal vertices). Furthermore, the breadth-first search is known to be inefficient for multiprocessor scheduling (Jonsson and Shin, 1997). So, LIFO strategy has been implemented.

• The vertex branching rule generates child vertices of the explored vertex and stores them in the set B. Two cases have to be considered according to the development of the current sub-tree. If there is some task without priority in the current sub-tree, then a vertex is generated to each of these tasks with the next priority level in the sub-tree. If every task has a priority, then the first priority level of the next processor is generated (eligible tasks to the lowest priority level). If the last processor is already considered, no new vertex is generated. The set B is sorted with the Deadline Monotonic rule (i.e. non-decreasing order of the relative deadlines). Thus with the depth-first search strategy, the first goal vertex reached is this one that assigns priorities according to the Deadline Monotonic scheduling rule for each processor or networks.

• The Lower Bound cost functions are used to estimate lower bounds of the worst-case response time of the tasks. For every explored vertex, a lower bound is computed for every task (having or not a priority). For that we have modified the holistic analysis in order to take into account tasks without priorities. The release jitter J_i of a task τ_i is the difference between the release of a task and its arrival in the system. It models the worst-case phasing of a task due to its incoming communications. This worst-case phasing for a task τ_i depends directly of the worst-case response times of its immediate predecessors, noted hereafter $Pred(i)$. Lower bounds (LB) of the worst-case response times are evaluated for fixed values of release jitters. After that, release jitters are updated according to the new values of the worst-case response times. This fundamental loop is done until two successive iterations lead to the same worst-case response time for every task. The system of recurrent equations associated to the vertex v of the search tree is:

$$1 \leq i \leq n \begin{cases} J_i^{(0)} &= 0 \\ R_i^{(k)} &= LB\left(i, J_i^{(k-1)}\right) \\ J_i^{(k)} &= \max\limits_{j \,\in\, pred(i)} \left(R_j^{(k)}\right) \end{cases} \qquad (1)$$

$$R_i = R_i^{(k)} = R_i^{(k-1)}$$

Obviously, the LB functions must be non decreasing to enforce the convergence of the whole system. These functions depend on the schedulers, that may be different from one processor to another. For instance, the processor modeling the CAN network uses a non-preemptive fixed priority

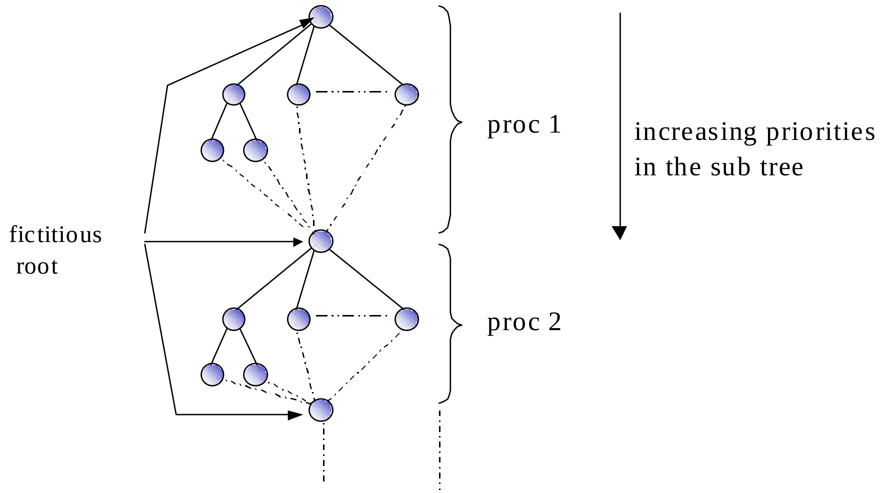

Fig. 2. Structure of the search tree

```
•    Algorithm Branch and Bound
1.   Initialise Active Set $A$ with vertices $1..n_1$ corresponding to tasks with lower priority level for the first processor
2.   While $A \neq \phi$ Loop
3.   Select a vertex in $A$ according to the vertex selection rule
4.   Generate a set $B$ of child vertices according to the vertex branching rule.
5.   Calculate the lower bounds for each vertex in $B$.
6.   Eliminate vertices in $B$ according the vertex elimination rule.
7.   If it exists a goal vertex in $B$ STOP
     Else moves all remaining vertices in $B$ to the active set $A$.
8.   End Loop
```

Table 2. Pseudo-code of the Branch&Bond algorithm

scheduler and a processing unit allows preemption for running tasks. We now successively detail the lower bounds for processing units and networks.

- Lower bounds of worst-case response times for processing units:
The current vertex defines a set of priorities for all its ancestors, but all the other tasks have not been assigned priority levels. Note that $[i]$ is the index of the tasks having the priority level i. Worst-case response time of a task is obtained in a processor busy period where all the higher priority tasks are simultaneously released at the beginning of the period. Such scenario defines the worst-case computational interference of higher priority tasks. At a given iteration of the system 1, J_i is a lower bound of the worst-case release jitter. According to this scenario, a lower bound of the worst-case workload of the processor running tasks with priority levels greater or equal than i in the interval $[0, t)$ is given by (Tindell and Clark, 1994):

$$W(t) = C_{[i]} + \sum_{j=1}^{i-1} \left\lceil \frac{t + J_{[j]}}{T_{[j]}} \right\rceil C_{[j]} \qquad (2)$$

If the task τ_i has no priority, then we cannot evaluate the interference of higher priority tasks since priority levels are assigned in increasing order in the search tree. So, a lower bound is obtained by assuming that the task τ_i is assigned the highest priority level. The lower bound of the workload in the interval $[0, t)$ is:

$$W(t) = C_{[i]} \qquad (3)$$

In both cases, the length L_i of the busy period is defined by a recurring equation that computes the length of the i-level busy period (Lehoczky, 1990). Computations stop when $L_i = L_i^{(k-1)} = L_i^{(k)}$.

$$\begin{cases} L_i^{(0)} & = \sum_{j=1}^{i} C_{[j]} \\ L_i^{(k+1)} & = W\left(L_i^{(k)}\right) \end{cases} \qquad (4)$$

The response time of τ_i, while considering the explored vertex, is: $R_i = J_i + L_i$.

- Lower bounds of worst-case response times for the networks:

The only difference is due to the non-preemptive dispatching strategy. Messages are scheduled using the fixed priorities defined in the explored vertex. Two cases must be considered according to whether the messages have been assigned a priority or not in the current vertex. The technique presented in the preemptive case can be applied for computing a lower bound of the worst-case response time of the messages, but it must be adapted in order to take into account the fact that a lower priority messages can be sent just before the beginning of the i-level busy period. This worst-case phasing is bounded to the longest message among lower priority ones. If n_p is the number of messages defined for the network number p, then the message with priority level i suffers interference from messages with higher or equal priority that is lower bounded by (Tindell *et al.*, 1995):

$$W(t) = C_{[i]} + \sum_{j=1}^{i-1} \left\lceil \frac{t + J_{[j]}}{T_{[j]}} \right\rceil C_{[j]} + \max_{k=i+1..n_p} (C_{[k]}) \qquad (5)$$

If the priority of a message number i is not known, a lower bound of its worst-case response time is obtained by considering that it should be assigned the highest priority level. That is to say:

$$W(t) = C_{[i]} + \max_{j=1..n_p, j\neq i} (C_j) \qquad (6)$$

These lower bounds can lead to the worst-case response time of the messages using the system of equations (4). As in the preemptive case, the worst-case response time is $R_i = J_i + L_i$, associated to the current explored vertex in the search tree. According to these lower bounds, we prove that the Branch&Bound method is optimal (in respect to holistic schedules). That is achieved if, and only if, the lower bounds of the worst-case response times are non-decreasing in every directed path in the search tree.

Property 1. Let $T = (V, E)$ be the search tree built by the branch and bound algorithm, V be the set of vertices and E be the set of edges, then $\forall a, b \in V^2$ such that $a \prec b$ in the search tree then $R_i^a \leq R_i^b, 1 \leq i \leq n$, where R_i^a (resp. R_i^b) denotes the lower bound of the worst-case response time of task τ_i while considering the vertex a (resp. b).

Proof: We consider an arbitrary path in the search tree, let a and b be two vertices of this path. Without loss of generality, we assume that $a \prec b$. We consider two cases according to whether a and b belong to the same sub-tree or not.
If a and b do not belong to the same sub-tree, then the response time R_i^b can only depend on the release jitters. The system (1) is used and enforces release jitters to be non-decreasing for every task. So lower bounds of the worst-case response times are also non-decreasing. So we observe that $R_i^a \leq R_i^b, 1 \leq i \leq n$.
If a and b belong to the same sub-tree, then vertices a and b model the priority assignment of tasks on the same processor. Without loss of generality, we consider an arbitrary task τ_i. If τ_i has a priority for vertices a and b, then worst-case response times can only change due to the increase of some release jitters. Since in (1) the LB functions are non-decreasing according to the release jitters, we verify that $R_i^a \leq R_i^b, 1 \leq i \leq n$. Now, if τ_i has a priority in vertex b and not in vertex a, then the lower bounds of τ_i are not computed using the same formula. In preemptive case, bounds (3) is necessarily lower than or equal to this one computed using (2). In non-preemptive case, if a task τ_k leading to a blocking effect in (6) is not the same than in (5), then τ_k have an higher priority than τ_i. So bounds computed by these computed by (6) is necessarily lower than or equal to (5). Finally, we verify that:

$$R_i^a \leq R_i^b, 1 \leq i \leq n \qquad (7)$$

• The vertex elimination rule is applied after computing the lower bounds of the worst-case response times of the child vertices of the explored vertex. A child vertex would lead to a feasible schedule (according to the holistic analysis performed on the goal vertices) if, and only if: $R_i \leq D_i, 1 \leq i \leq n$. If one task has a lower bound of its worst-case response time greater than its relative deadline, then no successor vertex will improve its response time since the lower bounds are non-decreasing functions (i.e., property 1). The considered child vertex can be pruned (deleted).

4. SOLVING EFFICIENCY

In order to prove the performance of the algorithm we have randomly generated workload for tasks and messages, while exactly respecting the global architecture presented in section 2 (i.e. number of tasks and messages, number of computers and networks, precedence relations between tasks and messages are definitely fixed). The parameters of each task or message have been randomly generated according the classical uniform law in order to reach a given workload of the processors and the networks, fixed by the utilization factors. Dependent tasks have been assigned the same period and end-to-end deadline. Because, if dependent tasks do not work at the same rat, sooner or later, the slowest task will miss its deadline. Thirty problems have been generated for each interval of workload of the processors and networks ([0.2-0.3] to [0.8-0.9]). The branch and bound was stopped after one hour of computation.

Figure 3a reports the number of problems (among the thirty generated in each workload interval) that leads to a feasible/unfeasible schedule or that has not been solved after one hour of computation. Furthermore, this chart shows that some systems have been shown schedulable with a workload belonging to the interval [0.5, 0.6] on each processor (site). The sufficient schedulability test performed in the branch and bound procedure does not lead to an overly pessimistic estimation of the behavior of the distributed application.

Figure 3b shows the average resolution times of the branch and bound algorithm. As it can be remarked, the worst-case computation times are obtained for intermediate workloads of processors and networks. These results show that the priority assignment problem is 'easy' to solve when the utilization factors are low or high on the networks or the processing units. But in the intermediate cases of workload, the method needs high computational time in order to find a feasible solution. It illustrates the interest of our branch and bound since optimality cannot be achieved using heuristic methods.

5. CONCLUSION

We have presented an optimal priority assignment for fixed-priority tasks and messages in a automotive computerized system. The architecture is based on multiple fieldbus networks connecting uniprocessor computation units. Tasks and messages are on-line scheduled according to fixed priorities. The priority assignment is performed with a branch and bound algorithm. Goal vertices are proved schedulable or not using the holistic analysis (i.e. a sufficient schedulability condition). The solution space is modelled with a tree explored using a depth-first search strategy. For every explored vertex, lower bounds of worst-case response times are computed . These lower bounds are used to prune unfeasible vertices (i.e., vertices that will never lead to a feasible holistic schedule). The lower bound scheme is based on the adaptation of the holistic analysis for tasks without a priori known priorities.

The perspectives of this work are first to extend the method to the mapping of task for multiprocessor computation units simultaneously with the calculation of the priorities for the tasks and messages on every processor or network. In second we want to take into account more complex relation among tasks as additional real-time traffic in the networks, communication of tasks working at different rates (generalized precedence constraints (Richard *et al.*, 2001)) and resource access protocol.

Last, we also want to compare our approach with known heuristic methods on the number of problems solved. It is an important issue, since when a method do not find a set of priorities, the system must be overdimension in order to meet its timing requirements.

6. REFERENCES

Buttazzo, G.C. (1997). *Hard real-time computing systems: predictable scheduling algorithms and applications.* Kluwer Academic Publishers.

Castelpietra, P., Y.Q. Song, F. Simonot-Lion and O. Cayrol (2000). Performance evalutation of a multiple networked in-vehicle embedded architecture. *proc. Workshop on Factory Communication Systems, Porto.*

Cavalieri, S., A Di-Stefano and O. Mirabella (1995). Pre-run-time scheduling to reduce schedule lenght in the fieldbus environment. *IEEE Transactions on Software Engineering* **21**, 865–880.

Cavalieri, S., A Di-Stefano and O. Mirabella (1996). Mappind automotive process control on iec/isa fieldbus functionalities. *Computers in industry* **28**, 233–250.

Dipippo, L.C. (2001). Scheduling and priority mapping for static real-time middleware. *J. Real-Time Time Systems* **20**, 155–182.

Gutierrez-Garcia, J.J. and M. Gonzalez-Harbour (1995). Optimized priority assignment for tasks and messages in distributed hard real-time systems. *proc. Workshop on Parallel and Distributed Hard Real-Time Systems, Santa Barbara.*

ISO (1994*a*). Road vehicles - low speed serial data communication - part 2: low-speed controller area network. *ISO 11519-2.*

ISO (1994*b*). Vehicle area network, serial data communication - road vehicle, serial data communication for automotive application. *ISO 11519-3.*

Jonsson, J. and K.G. Shin (1997). A parametrized branch and bound strategy for scheduling precedence-constrained tasks on a multiprocessor system. *proc. Int; Conf. on Parallel Processing (ICPP'97)* pp. 158–165.

Kopetz, H. (1997). *Hard real-time systems.* Kluwer Academic Publishers.

Lehoczky, J.P. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. *proc. IEEE Real-Time Systems Symposium.*

Liu, J.W.S. (2000). *Real-time systems.* Prentice Hall.

Lonn, H. and J. Axelsson (1999). A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. *proc. Euromicro on Real-Time Systems.*

Navet, N. and Y.Q. Song (1999). Reliability improvement of the dual priority protocol under unreliable transmission. *Control Engineering Practice* **7**, 975–981.

Norström, C., M. Gustafsson, J. Maki-Turja and N-E. Bånkestad (2000). Findings from introduding state of the art real-time technics in vehicle industry. *proc. Euromicro on Real-Time Systems, Stockohlm.*

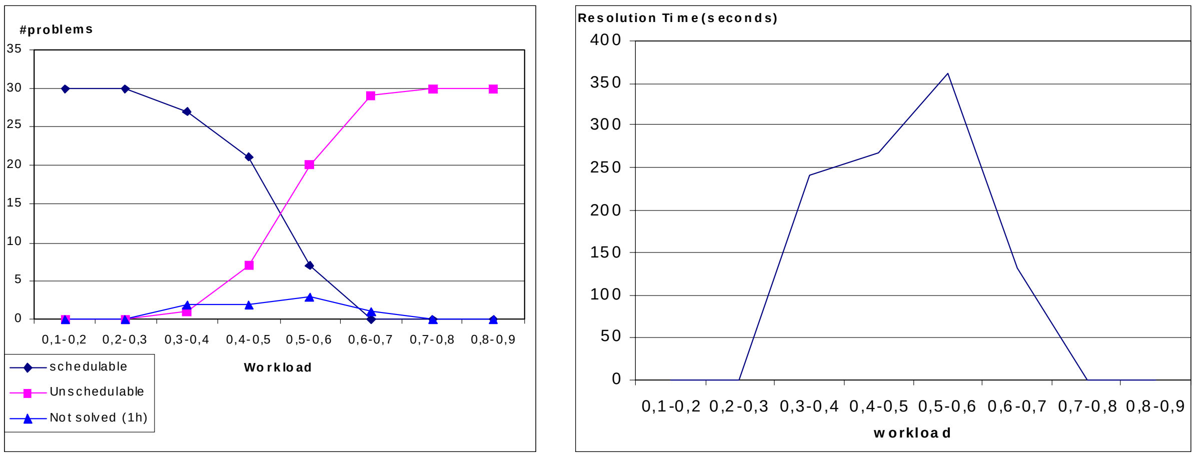

Fig. 3. Number of problems solved in 1 hour (3a) and Average resolution time in seconds (3b)

Osek (1997). Osek operating system, version 2.0r1. *http://www-iiit.etec.uni-karlsruhe.de/ osek/*.

Richard, P., M. Richard and F. Cottet (2001). On-line scheduling of real-time distributed computers with complex communication constraints. *proc. 7th IEEE Int. Conf. on Engineering of Complex Computer Systems* pp. 26–34.

Richard, P., M. Richard and F. Cottet (n.d.). Analyse holistique des systm̀es temps rél distribuś : principes et algorithmes. *Calculateurs Parallèles,Réseaux et Systèmes Répartis , to appear*.

Sandström, K., C. Eriksson and G. Fohler (1998). Handling interrupts with static scheduling in an automotive vehicle control system. *proc. Int. Conf. on Real-Time Computing Systems and Applications (RTCSA'98), Hiroshima (Japan)*.

Silva, R. and J. Fraga (2000). Fixed priority scheduling of tasks with arbitrary precedence consrtaints in distributed hard real-time systems. *Journal of System Architecture* **46**, 991–1004.

Sjödin, M. and H. Hansson (1998). Improved response time analysis calculation. *proc. IEEE Real-Time Systems Symposium*.

Song, Y.Q., F. Simonot-Lion and N. Navet (1999). De l'évaluation des performances du systèmes de communication à la validation de l'architecture opérationnelle - cas du sysème embarqué dans l'automobile. *Summer school on Real-Time Systems, Poitiers (France)*.

Thomesse, J.P. (1999). Fieldbuses and interoperability. *Control Engineering Practice* **7**, 81–94.

Tindell, K.W., A. Bruns and A.J. Wellings (1995). Calculating controller area network (can) message response times. *Control Engineering Practice* **3**, 1163–1169.

Tindell, K.W., A. Burns and A.J. Wellings (1992). Allocating hard real-time tasks: an np-hard problem made easy. *J. Real-Time Time Systems* **4**, 145–165.

Tindell, K.W. and J. Clark (1994). Holistic analysis for distributed hard real-time systems. *Microprocessors and Microprogramming* **40**, 117–134.